ECE 3663

Design Review 2

4/8/2014

Group OR

Jacob Breiholz

Emilio Esteban

Gabriel Ritter

**Design Review 2 Report**

At the end of the first design review we had completed the components for the 16 bit AND,OR, and PASS A as well as a 1 bit 8:1 mux. Our proposal for the arbitrary component was a 16 bit XNOR and we created a sketch for the ALU schematic. Now in addition, we have completed the 16 bit ADD, SUB, SHIFT, and Register. Furthermore we have now connected all of the current components together into the ALU, which also meant updating the 1 bit 8:1 mux into a 16 bit 8:1 mux.

The SHIFT can variably shift 1, 2, 3, or 4 bits to the left. This was done by using a barrel shifter composed of 2:1 muxes. Three columns of 16 muxes were used where the first column can shift one bit, the second can shift two bits, and the third can shift four bits. They can also shift any combination of 1, 2, and 4 although in this case we only care about three. The muxes shift by either passing in the corresponding bit input or a previous/lower input bit depending on how much is being shifted. If there is no previous/lower input the passes through a ‘0’. Additionally since there are three control points needed but two control inputs, some basic combinational logic is used A left shift was chosen because that is what was specified in the assignment.

The adder and subtractor were designed based off of the mirror adder we learned about in class. The mirror adder has a reasonable amount of delay and we were able to update an 8 bit design built for an assignment into a 16 bit one. The subtractor was created by inverting one of the 16 bit inputs and tying the carry in high instead of low. This process converts one of the inputs into a 2’s complement number. Adding a positive number and a negative 2’s compliment number results in subtraction.

The memory register was constructed from two latches in series, one positive level sensitive and the other negative level sensitive. The register was constructed to be triggered by rising edges, which was an arbitrary but generally more used choice. The simulations included show the operation of an individual register. Since the 16 bit register is essentially several independent 1 bit registers in parallel we decided against completely simulating the 16 bit implementation. However, we did include a simulation with the lowest 3 bits of the device for completeness.

For our finalization of the arbitrary function, the XNOR was determined to be too simplistic so in the end a multiplier was decided on. The multiplier will take the least significant 8 bits of the A and B inputs to the ALU and then multiply them out to 16 bits using a Wallace Tree multiplier. The remaining most significant bits of the two inputs will be discarded. A Wallace Tree multiplier is chosen because of its speed compared to the array multiplier as we learned about in class. A four stage compression is sufficient to turn the partial products into two numbers that can be used by a fast adder. We have included a diagram of one possible way to implement the Wallace tree compression on the next page.

NOP was implemented by routing the output of the output mux back into the mux controlled by the select lines, so that when NOP occurs, the output remains the same.

Finally, we wired everything together to create a functional ALU. The following show select inputs show the commands:

|  |  |
| --- | --- |
| Function | Control (“S2,S1,S0”) |
| ADD | “000” |
| SUB | “001” |
| SHIFT | “010” |
| AND | “011” |
| OR | “100” |
| PASS | “101” |
| Arbitrary | “110” |
| NOP | “111” |

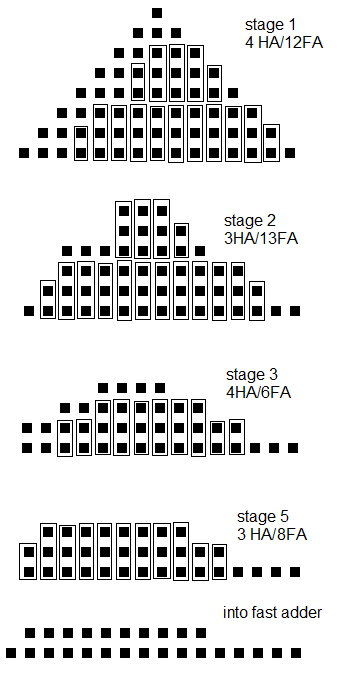
The next steps for this project involve the addition of multiplier into our ALU and perhaps the combining of the adder and subtractor for reduced components. This reduction would involve muxes to the inputs to either invert or keep the signal the same and a mux to the carry in to either input one or zero. Subtraction will work by adding a 2’s complement number (inverted input plus the carry in).

Additionally we will use logical effort to calculate the optimum sizings and from there we can determine the delay and therefore optimum clock cycle.

Our final task will be to calculate metrics such as delay and power. Once this is done all that remains is the preparation of the paper and presentation.

**Wallace Tree Compression**

In the diagram below, each block represents a one-bit partial product. the blocks are circled in groups of two (half-adders) and three (full-adders) which will produce an equivalent partial product of one number in the same column and another number carried to the next most significant column. The carries are not shared within a stage, therefore the critical path will only pass through one adder at most for each stage.

Note also that many of the lower partial products get compressed into one number, bypassing the need to include them in the fast adder at all. The four compression stages proposed use a total of 14 half adders and 39 full adders, and will then feed into a fast adder that need only consist of 9 full adders and a half adder on each end.

**Component Delays:**

|  |  |  |
| --- | --- | --- |
| Operation | Worst-Case-Delay (seconds) | Scenario (for tpLH and tpHL) |
| ADD | 9.495\*10-10 | Cin:0->1,S15:1->0, and  Cin:1->, S15:0->1 |
| SUB | 4.945\*10-10 | B0:0->1,S15:1->0 and  B0:1->0,S15:0->1 |
| SHIFT | 3.54\*10-10 | Cntl1:1,Cntl2:0->1,In12:1->0 and  Cntl1:1,Cntl2:1->0,In13:1->0 |
| AND | 1.4395\*10-11 | A:0->1,B:1 and A:1->0,B:1 |
| OR | 1.4555\*10-11 | A:0->1,B:1 and A:1->0,B:1 |
| PASS A | 1.5119\*10-11 | A: 0->1 and 1->0 |

**Schematics**

Figure 1 - Adder 16b

This schematic is hierarchical; the lower level schematics are shown below. The 8b adder is shown in figure 2, the 1b inverter is shown in figure 4.

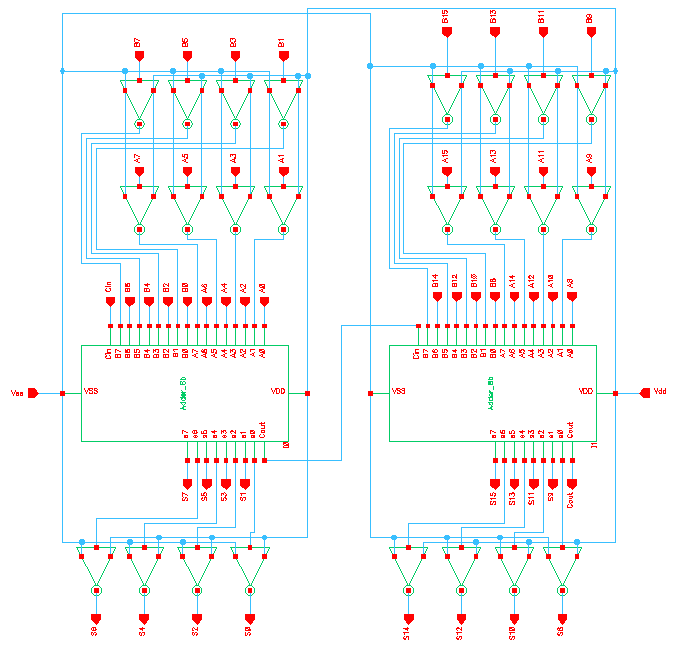
****

Figure 2 – Adder 8b

This schematic is hierarchical; the lower level schematic is shown below. The 1b mirror adder is shown in figure 3.

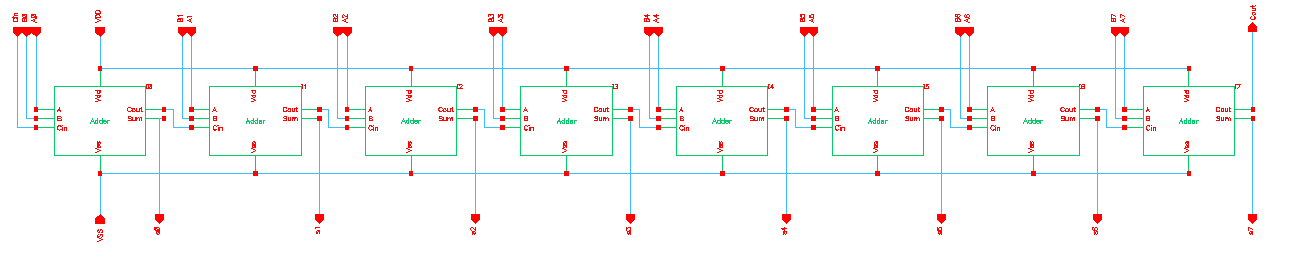
****

Figure 3 – Mirror Adder 1b

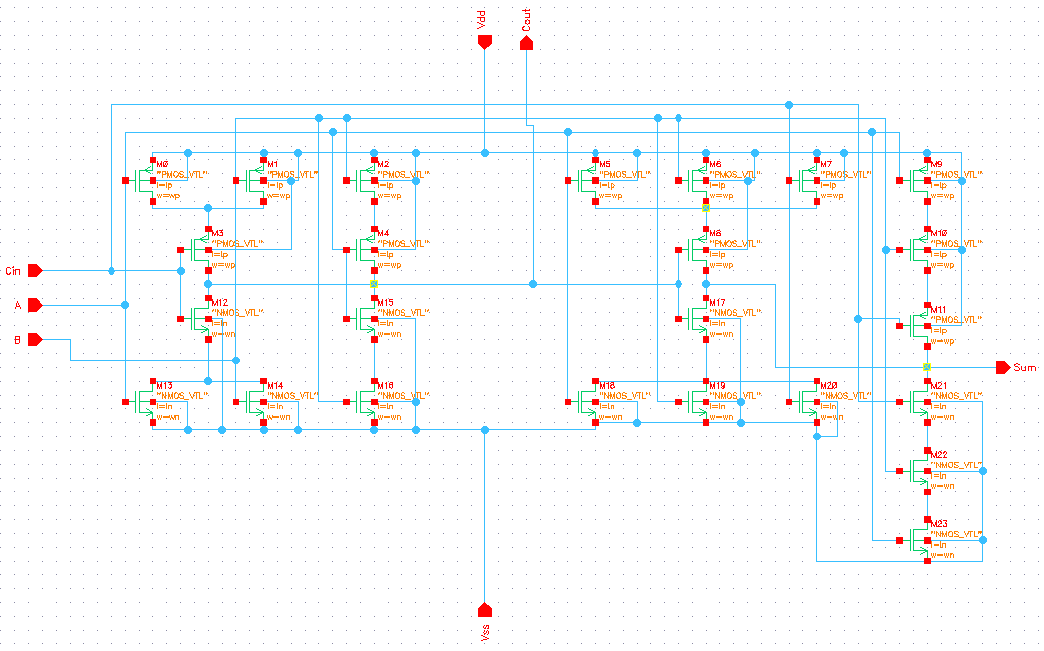
****

Figure 4 – Inverter 1b

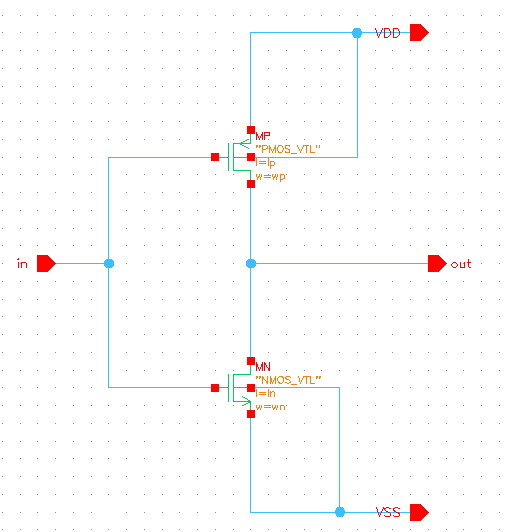
****

Figure 5 – Subtractor 16b

This schematic is hierarchical; the lower level schematics is are shown above. The 1b inverters are shown in figure 4, and the 8 bit adders are shown in figure 2.

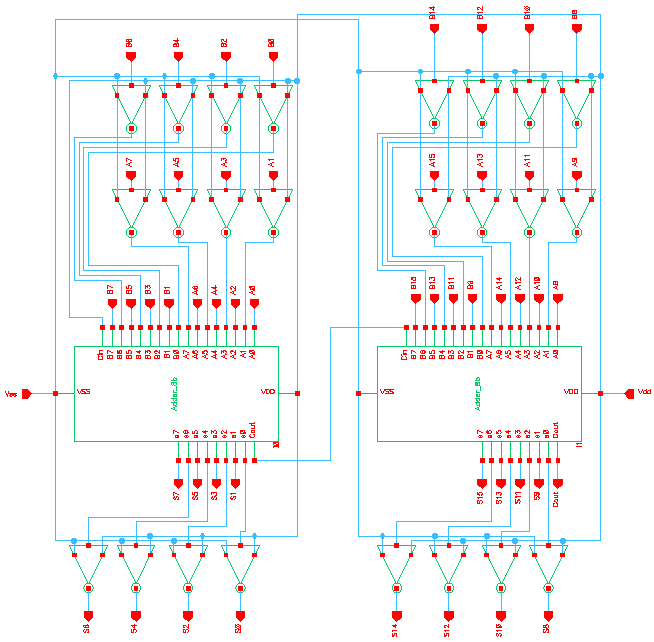
****

Figure 6 – Register 16b

This schematic is hierarchical; the lower level schematics are shown below. The 1b register is shown in figure 7.

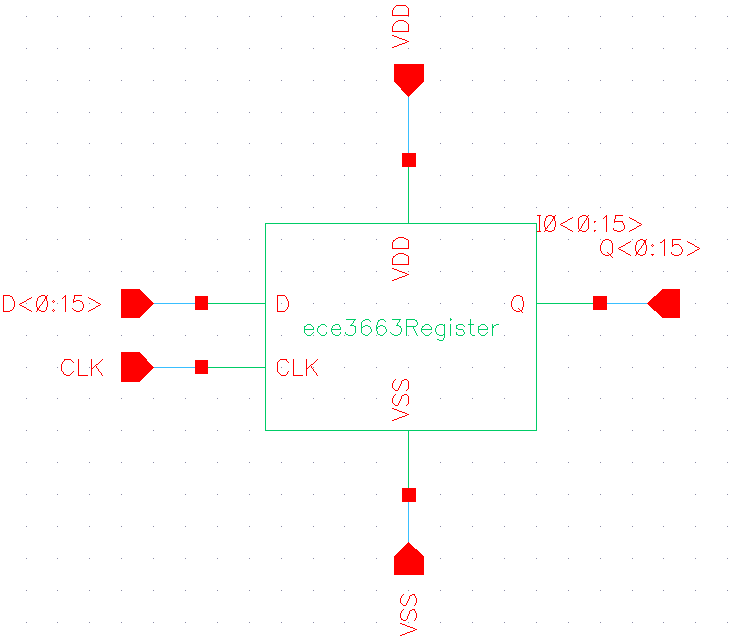


Figure 7 – Register 1b

This schematic is hierarchical; the lower level schematics are shown below. The 1b latches are shown in figure 8.

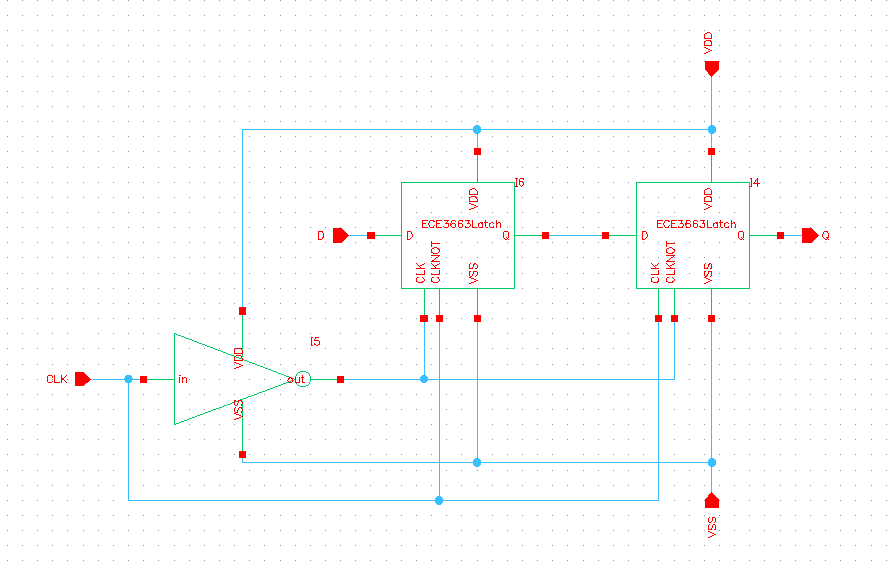
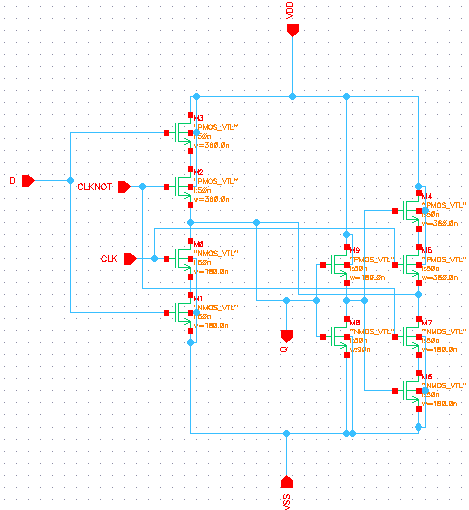


Figure 8 – Latch 1b



**We decided to create the 16b shifter directly (via netlist), so we do not have any schematics for it. The netlist is shown below.**

//16 Bit shifter : Input 16b Control 2b. “00” 1b shift, “01” 2b shift, “10” 3b shift, “11” 4b shift

//netlist. No schematic was created for this, it was entirely done using a netlist

include "/app/lib/freepdk45/trunk/ncsu\_basekit/models/hspice/tran\_models/models\_nom/NMOS\_VTL.inc"

include "/app/lib/freepdk45/trunk/ncsu\_basekit/models/hspice/tran\_models/models\_nom/PMOS\_VTL.inc"

subckt Inverter VDD VSS in out

parameters wp=90n wn=90n ln=50n lp=50n mult=1

// (d g s b)

MN (out in VSS VSS) NMOS\_VTL w=wn l=ln as=100e-9\*wn ad=100e-9\*wn \

ps=200e-9+wn pd=200e-9+wn ld=105n ls=105n m=mult

MP (out in VDD VDD) PMOS\_VTL w=wp l=lp as=100e-9\*wp ad=100e-9\*wp \

ps=200e-9+wp pd=200e-9+wp ld=105n ls=105n m=mult

ends Inverter

// NAND GATE

subckt NAND (VDD VSS A B out)

parameters wp=90n wn=180n lp=50n ln=50n mult=1

M0 (out A VDD VDD) PMOS\_VTL w=wp l=lp as=100n\*wp ad=100n\*wp ps=200n+wp \

pd=200n+wp m=mult

M1 (out B VDD VDD) PMOS\_VTL w=wp l=lp as=100n\*wp ad=100n\*wp ps=200n+wp \

pd=200n+wp m=mult

M2 (net5 B VSS VSS) NMOS\_VTL w=wn l=ln as=100n\*2\*wn ad=100n\*2\*wn ps=200n+2\*wn \

pd=200n+2\*wn m=mult

M3 (out A net5 VSS) NMOS\_VTL w=wn l=ln as=100n\*2\*wn ad=100n\*2\*wn ps=200n+2\*wn \

pd=200n+2\*wn m=mult

ends NAND

//XNOR GATE

subckt XNOR (VDD VSS A B out)

parameters wp=90n wn=90n ln=50n lp=50n mult=1

M0 (out A net0 VSS) NMOS\_VTL w=wn\*2 l=lp as=100n\*wn\*2 ad=100n\*wn\*2 ps=200n+wn\*2 pd=200n+wn\*2 m=mult

M1 (net0 net4 VSS VSS) NMOS\_VTL w=wn\*2 l=lp as=100n\*wn\*2 ad=100n\*wn\*2 ps=200n+wn\*2 pd=200n+wn\*2 m=mult

M2 (out B net1 VSS) NMOS\_VTL w=wn\*2 l=lp as=100n\*wn\*2 ad=100n\*wn\*2 ps=200n+wn\*2 pd=200n+wn\*2 m=mult

M3 (net1 net3 VSS VSS) NMOS\_VTL w=wn\*2 l=lp as=100n\*wn\*2 ad=100n\*wn\*2 ps=200n+wn\*2 pd=200n+wn\*2 m=mult

M4 (net2 B VDD VDD) PMOS\_VTL w=wp\*2 l=lp as=100n\*wp\*2 ad=100n\*wp\*2 ps=200n+wp\*2 pd=200n+wp\*2 m=mult

M5 (out net4 net2 VDD) PMOS\_VTL w=wp\*2 l=lp as=100n\*wp\*2 ad=100n\*wp\*2 ps=200n+wp\*2 pd=200n+wp\*2 m=mult

M6 (net2 net3 VDD VDD) PMOS\_VTL w=wp\*2 l=lp as=100n\*wp\*2 ad=100n\*wp\*2 ps=200n+wp\*2 pd=200n+wp\*2 m=mult

M7 (out A net2 VDD) PMOS\_VTL w=wp\*2 l=lp as=100n\*wp\*2 ad=100n\*wp\*2 ps=200n+wp\*2 pd=200n+wp\*2 m=mult

IA (VDD VSS A net3) Inverter

IB (VDD VSS B net4) Inverter

ends XNOR

//2:1 MUX 1bit

subckt mux (Cnt In1 In2 OUT VDD VSS)

M2 (net7 Cnt VSS VSS) NMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

M1 (In2 Cnt OUT VSS) NMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

M0 (In1 net7 OUT VSS) NMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

M5 (net7 Cnt VDD VDD) PMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

M4 (In2 net7 OUT VDD) PMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

M3 (In1 Cnt OUT VDD) PMOS\_VTL w=90n l=50n as=9.45e-15 ad=9.45e-15 \

ps=300n pd=300n ld=105n ls=105n m=1

ends mux

//shifter

subckt shifter (Cntl1 Cntl2 VSS VDD In1 In2 In3 In4 In5 In6 In7 In8 In9 In10 In11 In12 In13 In14 In15 In16 Out1 Out2 Out3 Out4 Out5 Out6 Out7 Out8 Out9 Out10 Out11 Out12 Out13 Out14 Out15 Out16)

I1\_1 (Cntl1 VSS In1 out1\_1 VDD VSS) mux

I1\_2 (Cntl1 In1 In2 out1\_2 VDD VSS) mux

I1\_3 (Cntl1 In2 In3 out1\_3 VDD VSS) mux

I1\_4 (Cntl1 In3 In4 out1\_4 VDD VSS) mux

I1\_5 (Cntl1 In4 In5 out1\_5 VDD VSS) mux

I1\_6 (Cntl1 In5 In6 out1\_6 VDD VSS) mux

I1\_7 (Cntl1 In6 In7 out1\_7 VDD VSS) mux

I1\_8 (Cntl1 In7 In8 out1\_8 VDD VSS) mux

I1\_9 (Cntl1 In8 In9 out1\_9 VDD VSS) mux

I1\_10 (Cntl1 In9 In10 out1\_10 VDD VSS) mux

I1\_11 (Cntl1 In10 In11 out1\_11 VDD VSS) mux

I1\_12 (Cntl1 In11 In12 out1\_12 VDD VSS) mux

I1\_13 (Cntl1 In12 In13 out1\_13 VDD VSS) mux

I1\_14 (Cntl1 In13 In14 out1\_14 VDD VSS) mux

I1\_15 (Cntl1 In14 In15 out1\_15 VDD VSS) mux

I1\_16 (Cntl1 In15 In16 out1\_16 VDD VSS) mux

I2\_1 (cntmux2 VSS out1\_1 out2\_1 VDD VSS) mux

I2\_2 (cntmux2 VSS out1\_2 out2\_2 VDD VSS) mux

I2\_3 (cntmux2 out1\_1 out1\_3 out2\_3 VDD VSS) mux

I2\_4 (cntmux2 out1\_2 out1\_4 out2\_4 VDD VSS) mux

I2\_5 (cntmux2 out1\_3 out1\_5 out2\_5 VDD VSS) mux

I2\_6 (cntmux2 out1\_4 out1\_6 out2\_6 VDD VSS) mux

I2\_7 (cntmux2 out1\_5 out1\_7 out2\_7 VDD VSS) mux

I2\_8 (cntmux2 out1\_6 out1\_8 out2\_8 VDD VSS) mux

I2\_9 (cntmux2 out1\_7 out1\_9 out2\_9 VDD VSS) mux

I2\_10 (cntmux2 out1\_8 out1\_10 out2\_10 VDD VSS) mux

I2\_11 (cntmux2 out1\_9 out1\_11 out2\_11 VDD VSS) mux

I2\_12 (cntmux2 out1\_10 out1\_12 out2\_12 VDD VSS) mux

I2\_13 (cntmux2 out1\_11 out1\_13 out2\_13 VDD VSS) mux

I2\_14 (cntmux2 out1\_12 out1\_14 out2\_14 VDD VSS) mux

I2\_15 (cntmux2 out1\_13 out1\_15 out2\_15 VDD VSS) mux

I2\_16 (cntmux2 out1\_14 out1\_16 out2\_16 VDD VSS) mux

I3\_1 (cntmux3 VSS out2\_1 Out1 VDD VSS) mux

I3\_2 (cntmux3 VSS out2\_2 Out2 VDD VSS) mux

I3\_3 (cntmux3 VSS out2\_3 Out3 VDD VSS) mux

I3\_4 (cntmux3 VSS out2\_4 Out4 VDD VSS) mux

I3\_5 (cntmux3 out2\_1 out2\_5 Out5 VDD VSS) mux

I3\_6 (cntmux3 out2\_2 out2\_6 Out6 VDD VSS) mux

I3\_7 (cntmux3 out2\_3 out2\_7 Out7 VDD VSS) mux

I3\_8 (cntmux3 out2\_4 out2\_8 Out8 VDD VSS) mux

I3\_9 (cntmux3 out2\_5 out2\_9 Out9 VDD VSS) mux

I3\_10 (cntmux3 out2\_6 out2\_10 Out10 VDD VSS) mux

I3\_11 (cntmux3 out2\_7 out2\_11 Out11 VDD VSS) mux

I3\_12 (cntmux3 out2\_8 out2\_12 Out12 VDD VSS) mux

I3\_13 (cntmux3 out2\_9 out2\_13 Out13 VDD VSS) mux

I3\_14 (cntmux3 out2\_10 out2\_14 Out14 VDD VSS) mux

I3\_15 (cntmux3 out2\_11 out2\_15 Out15 VDD VSS) mux

I3\_16 (cntmux3 out2\_12 out2\_16 Out16 VDD VSS) mux

xnor1 (VDD VSS Cntl1 Cntl2 cntmux2) XNOR

nand1 (VDD VSS Cntl1 Cntl2 cntmux3) NAND

ends shifter

**We also decided to do the ALU level connections directly (via netlist), so we do not have any schematics for it. The relevant part of the netlist is shown below (because the full netlist was approaching 1000 lines, we left out the sub circuit definitions here).**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALU Functions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

myAdder (A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 AdderCarryin AdderCout AdderSum0 AdderSum1 AdderSum2 AdderSum3 AdderSum4 AdderSum5 AdderSum6 AdderSum7 AdderSum8 AdderSum9 AdderSum10 AdderSum11 AdderSum12 AdderSum13 AdderSum14 AdderSum15 VDD VSS) Adder\_16b

mySubtractor (A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 SubtractorCout SubtractorSum0 SubtractorSum1 SubtractorSum2 SubtractorSum3 SubtractorSum4 SubtractorSum5 SubtractorSum6 SubtractorSum7 SubtractorSum8 SubtractorSum9 SubtractorSum10 SubtractorSum11 SubtractorSum12 SubtractorSum13 SubtractorSum14 SubtractorSum15 VDD VSS) Subtractor\_16b

myAND (A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 VDD VSS AndOut0 AndOut1 AndOut2 AndOut3 AndOut4 AndOut5 AndOut6 AndOut7 AndOut8 AndOut9 AndOut10 AndOut11 AndOut12 AndOut13 AndOut14 AndOut15) ece3663AND\_16b

myOR (A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 VDD VSS OrOut0 OrOut1 OrOut2 OrOut3 OrOut4 OrOut5 OrOut6 OrOut7 OrOut8 OrOut9 OrOut10 OrOut11 OrOut12 OrOut13 OrOut14 OrOut15) ece3663OR\_16b

myPASSA (A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 VDD VSS PassAOut0 PassAOut1 PassAOut2 PassAOut3 PassAOut4 PassAOut5 PassAOut6 PassAOut7 PassAOut8 PassAOut9 PassAOut10 PassAOut11 PassAOut12 PassAOut13 PassAOut14 PassAOut15) ece3663PASSA\_16b

myShifter (B0 B1 VSS VDD A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 ShifterOut0 ShifterOut1 ShifterOut2 ShifterOut3 ShifterOut4 ShifterOut5 ShifterOut6 ShifterOut7 ShifterOut8 ShifterOut9 ShifterOut10 ShifterOut11 ShifterOut12 ShifterOut13 ShifterOut14 ShifterOut15) shifter

//arbitrary function

//outs ArbitOut0 - ArbitOut15

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Muxs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

myMux0 (AdderSum0 SubtractorSum0 ShifterOut0 AndOut0 OrOut0 PassAOut0 ArbitOut0 Out0 VDD VSS MuxOut0 S0 S1 S2) mux

myMux1 (AdderSum1 SubtractorSum1 ShifterOut1 AndOut1 OrOut1 PassAOut1 ArbitOut1 Out1 VDD VSS MuxOut1 S0 S1 S2) mux

myMux2 (AdderSum2 SubtractorSum2 ShifterOut2 AndOut2 OrOut2 PassAOut2 ArbitOut2 Out2 VDD VSS MuxOut2 S0 S1 S2) mux

myMux3 (AdderSum3 SubtractorSum3 ShifterOut3 AndOut3 OrOut3 PassAOut3 ArbitOut3 Out3 VDD VSS MuxOut3 S0 S1 S2) mux

myMux4 (AdderSum4 SubtractorSum4 ShifterOut4 AndOut4 OrOut4 PassAOut4 ArbitOut4 Out4 VDD VSS MuxOut4 S0 S1 S2) mux

myMux5 (AdderSum5 SubtractorSum5 ShifterOut5 AndOut5 OrOut5 PassAOut5 ArbitOut5 Out5 VDD VSS MuxOut5 S0 S1 S2) mux

myMux6 (AdderSum6 SubtractorSum6 ShifterOut6 AndOut6 OrOut6 PassAOut6 ArbitOut6 Out6 VDD VSS MuxOut6 S0 S1 S2) mux

myMux7 (AdderSum7 SubtractorSum7 ShifterOut7 AndOut7 OrOut7 PassAOut7 ArbitOut7 Out7 VDD VSS MuxOut7 S0 S1 S2) mux

myMux8 (AdderSum8 SubtractorSum8 ShifterOut8 AndOut8 OrOut8 PassAOut8 ArbitOut8 Out8 VDD VSS MuxOut8 S0 S1 S2) mux

myMux9 (AdderSum9 SubtractorSum9 ShifterOut9 AndOut9 OrOut9 PassAOut9 ArbitOut9 Out9 VDD VSS MuxOut9 S0 S1 S2) mux

myMux10 (AdderSum10 SubtractorSum10 ShifterOut10 AndOut10 OrOut10 PassAOut10 ArbitOut10 Out10 VDD VSS MuxOut10 S0 S1 S2) mux

myMux11 (AdderSum11 SubtractorSum11 ShifterOut11 AndOut11 OrOut11 PassAOut11 ArbitOut11 Out11 VDD VSS MuxOut11 S0 S1 S2) mux

myMux12 (AdderSum12 SubtractorSum12 ShifterOut12 AndOut12 OrOut12 PassAOut12 ArbitOut12 Out12 VDD VSS MuxOut12 S0 S1 S2) mux

myMux13 (AdderSum13 SubtractorSum13 ShifterOut13 AndOut13 OrOut13 PassAOut13 ArbitOut13 Out13 VDD VSS MuxOut13 S0 S1 S2) mux

myMux14 (AdderSum14 SubtractorSum14 ShifterOut14 AndOut14 OrOut14 PassAOut14 ArbitOut14 Out14 VDD VSS MuxOut14 S0 S1 S2) mux

myMux15 (AdderSum15 SubtractorSum15 ShifterOut15 AndOut15 OrOut15 PassAOut15 ArbitOut15 Out15 VDD VSS MuxOut15 S0 S1 S2) mux

//for Cout bits

//passes AdderCout if adder selected, SubtractorCout if subtractor selected, and VSS if anything else if selected

myMux16 (AdderCout SubtractorCout VSS VSS VSS VSS VSS VSS VDD VSS MuxCout S0 S1 S2) mux

InvLoad (VDD1 VSS1 MuxCout outload) Inverter

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Registers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

myRegisterOut (VDD VSS CLK MuxOut0 MuxOut1 MuxOut2 MuxOut3 MuxOut4 MuxOut5 MuxOut6 MuxOut7 MuxOut8 MuxOut9 MuxOut10 MuxOut11 MuxOut12 MuxOut13 MuxOut14 MuxOut15 Out0 Out1 Out2 Out3 Out4 Out5 Out6 Out7 Out8 Out9 Out10 Out11 Out12 Out13 Out14 Out15) ece3663Register\_16b

myRegisterA (VDD VSS CLK Ain0 Ain1 Ain2 Ain3 Ain4 Ain5 Ain6 Ain7 Ain8 Ain9 Ain10 Ain11 Ain12 Ain13 Ain14 Ain15 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15) ece3663Register\_16b

myRegisterB (VDD VSS CLK Bin0 Bin1 Bin2 Bin3 Bin4 Bin5 Bin6 Bin7 Bin8 Bin9 Bin10 Bin11 Bin12 Bin13 Bin14 Bin15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15) ece3663Register\_16b

**Simulations**

Figure 9 – Adder 16b Simulation 1

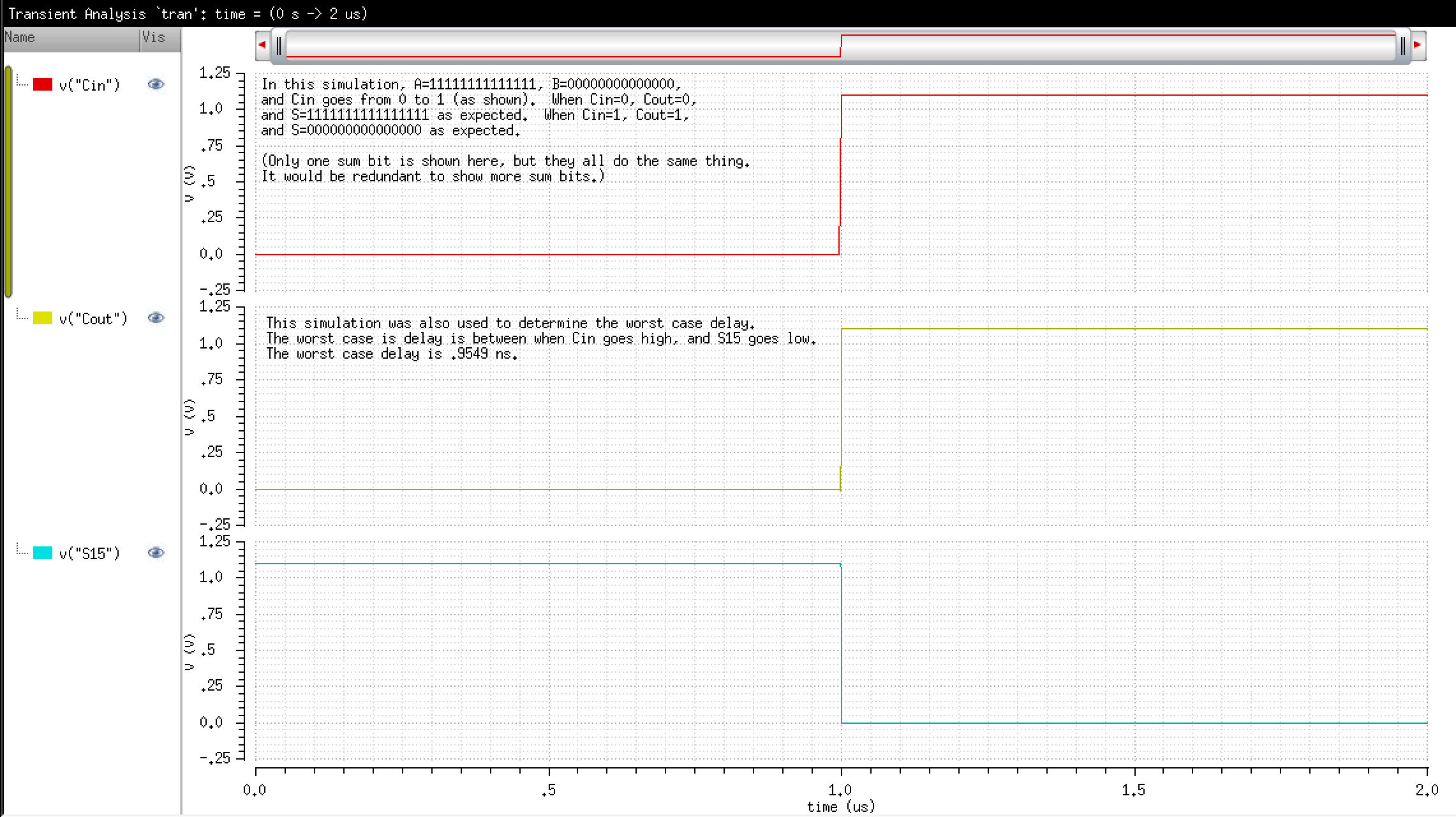


Figure 10 – Adder 16b Simulation 2

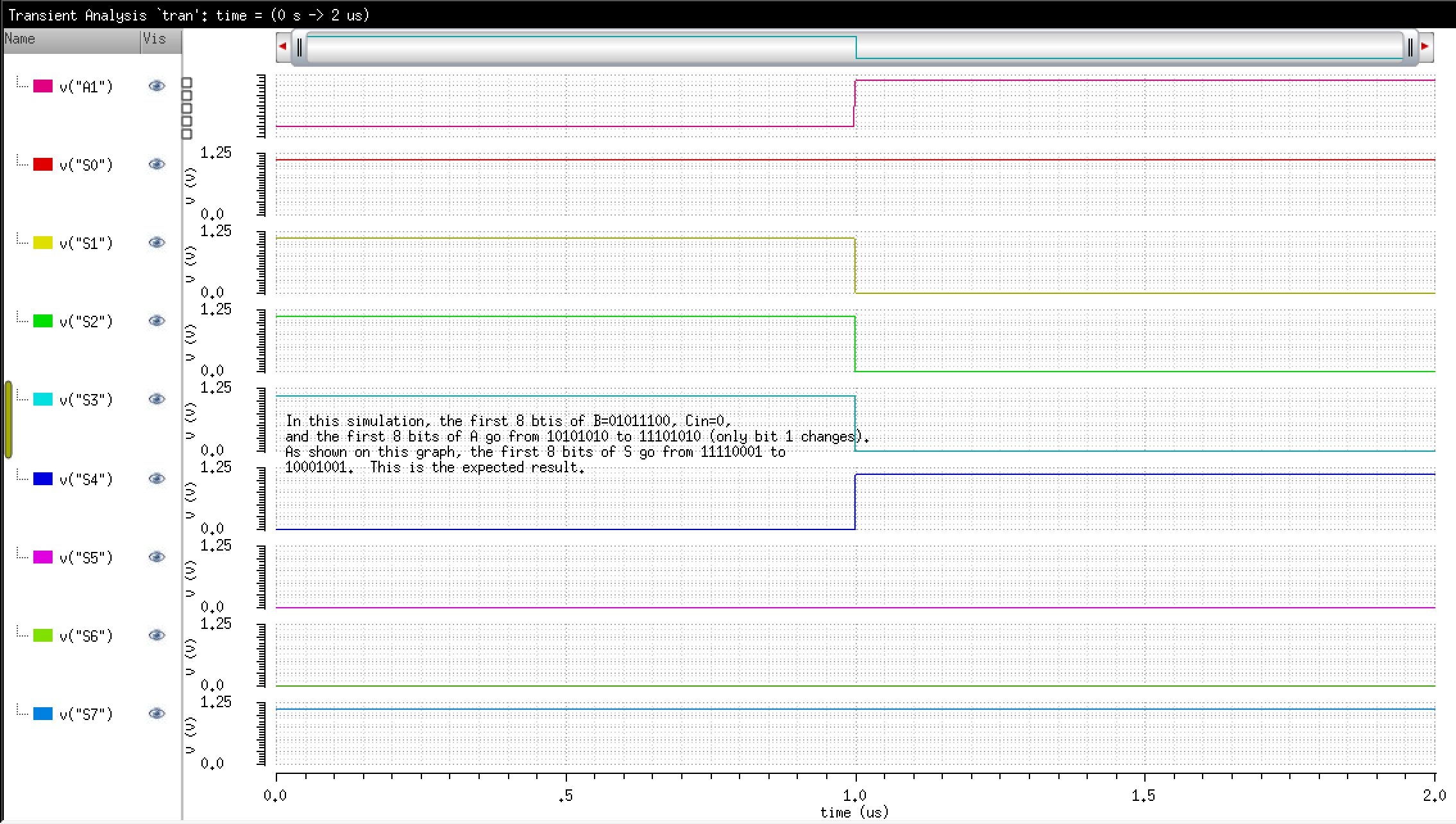


Figure 11 – Subtractor 16b Simulation 1

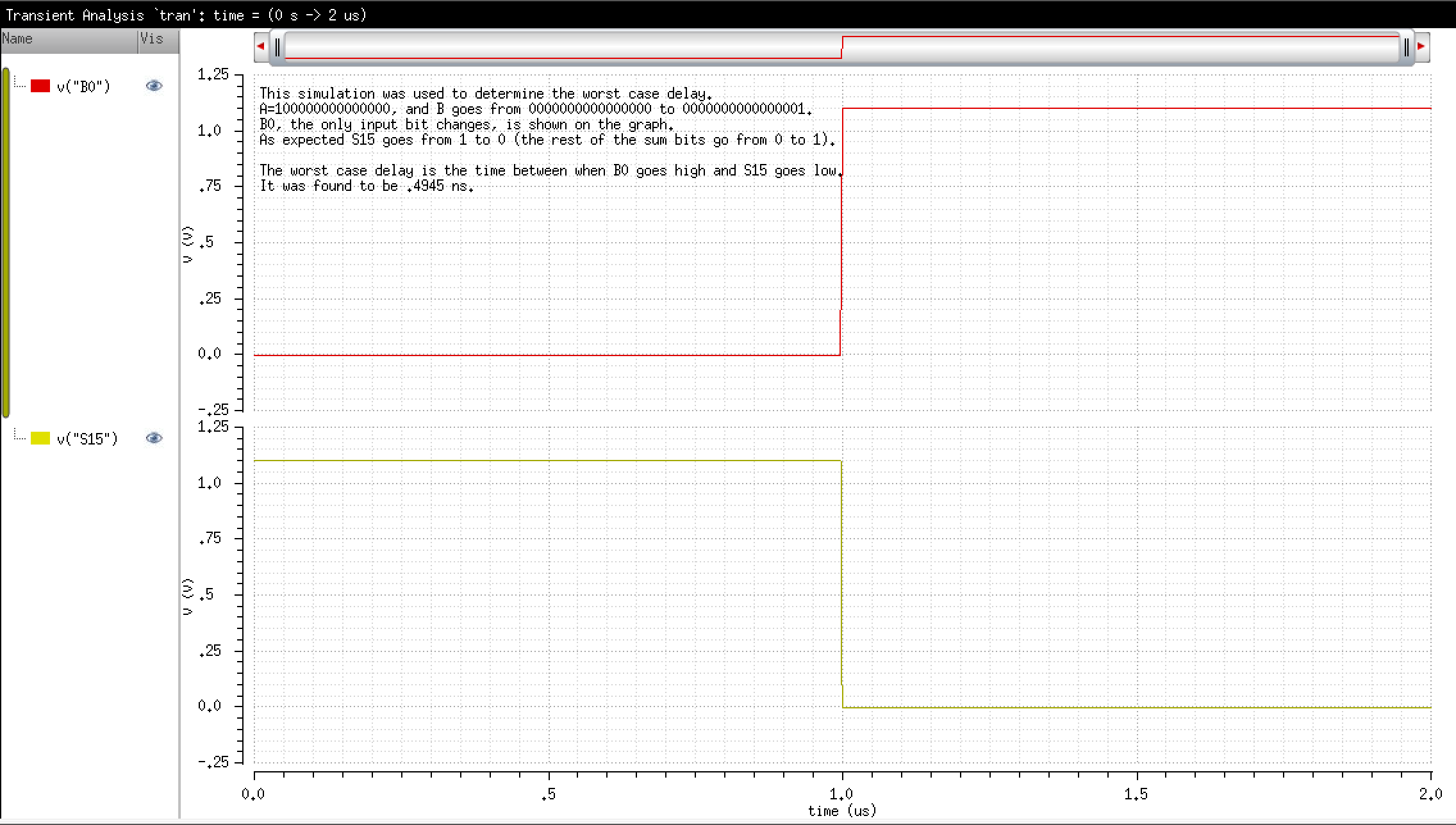


Figure 12 – Subtractor Simulation 2

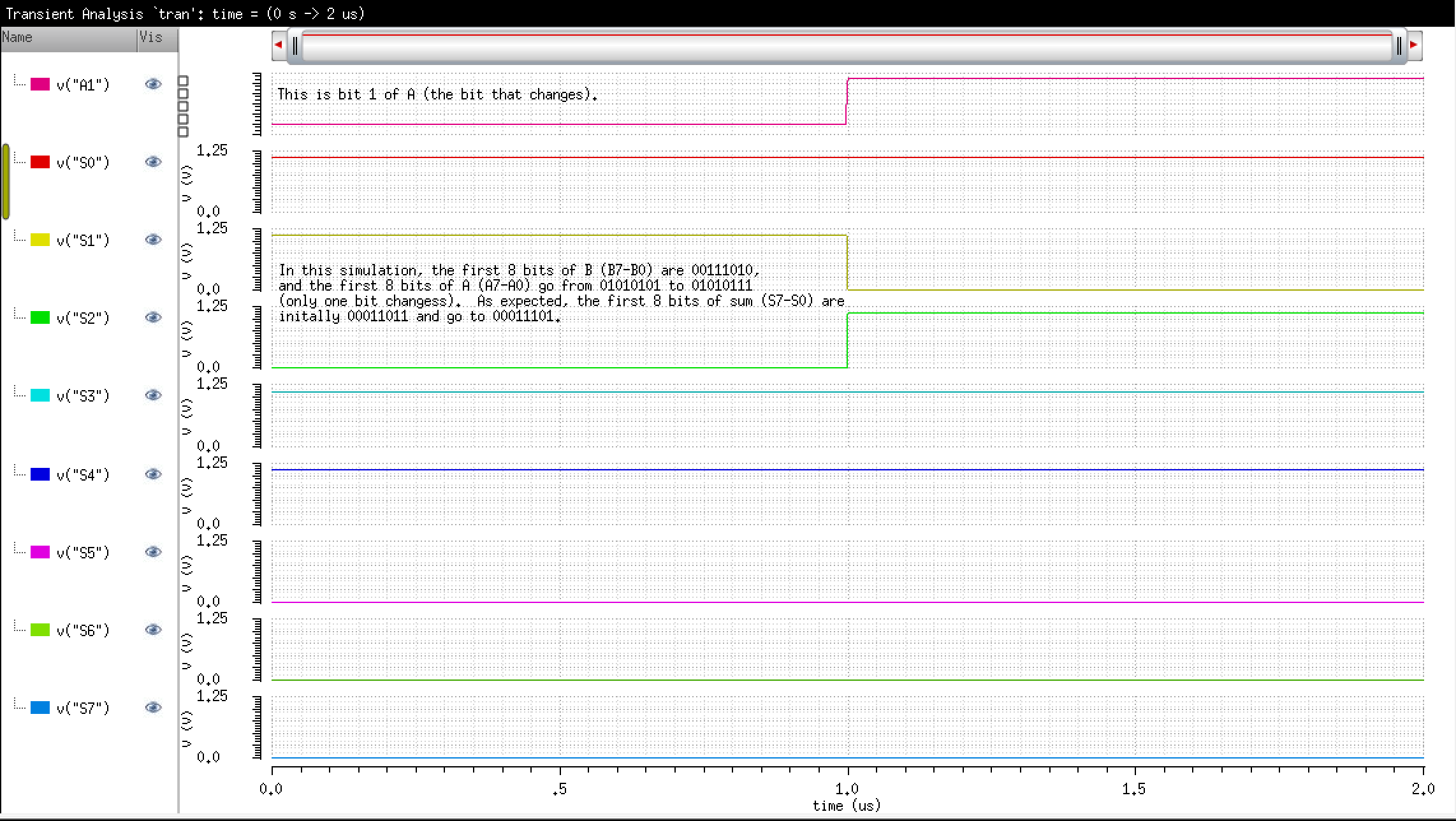
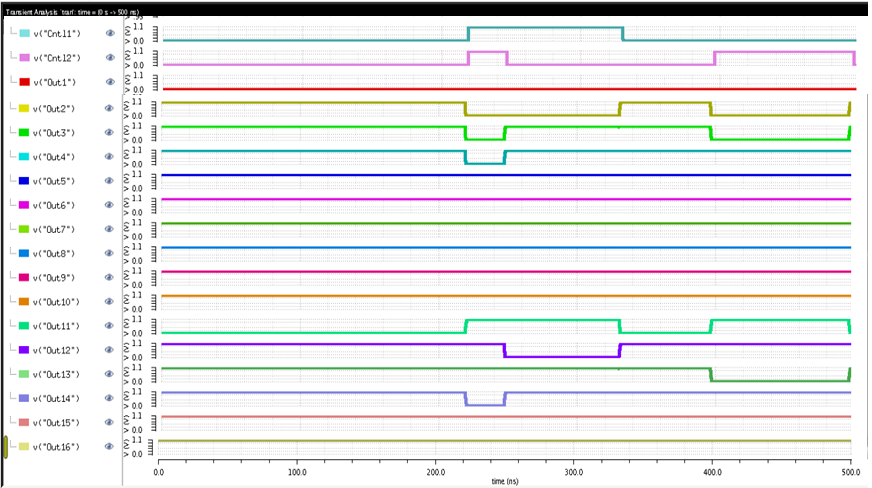


Figure 13 – 16b Shifter Simulation



The inputs for this plot are (msb) 1111101111111111 (lsb) and sel=Cntl2\_Cntl1.

The output of the 1st bit should always be 0 since it is always being shifted.

The output of the 2nd bit should be 1 only when there is 1 shift (sel=”00”)

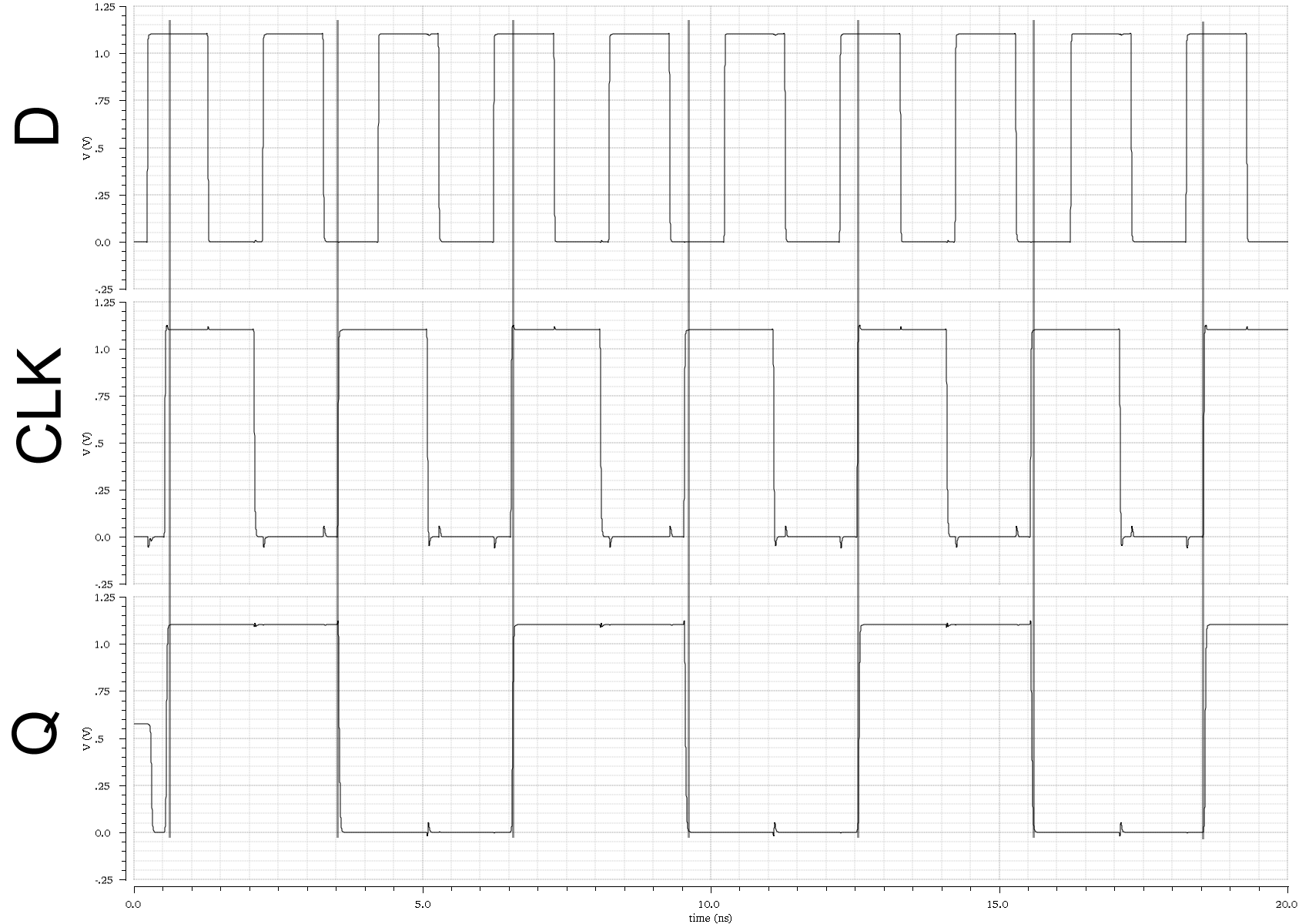
The output of the 3rd bit should be 1 when there is 1 or 2 shifts (sel=”00” or sel=”01”)

The output of the 4th bit should be 1 when there is 1, 2, or 3 shifts (00, 01, 10)

The output of the 5th through 10th bits will always be 1 since they will always be shifted in a 1 from the previous 1 through 4 inputs.

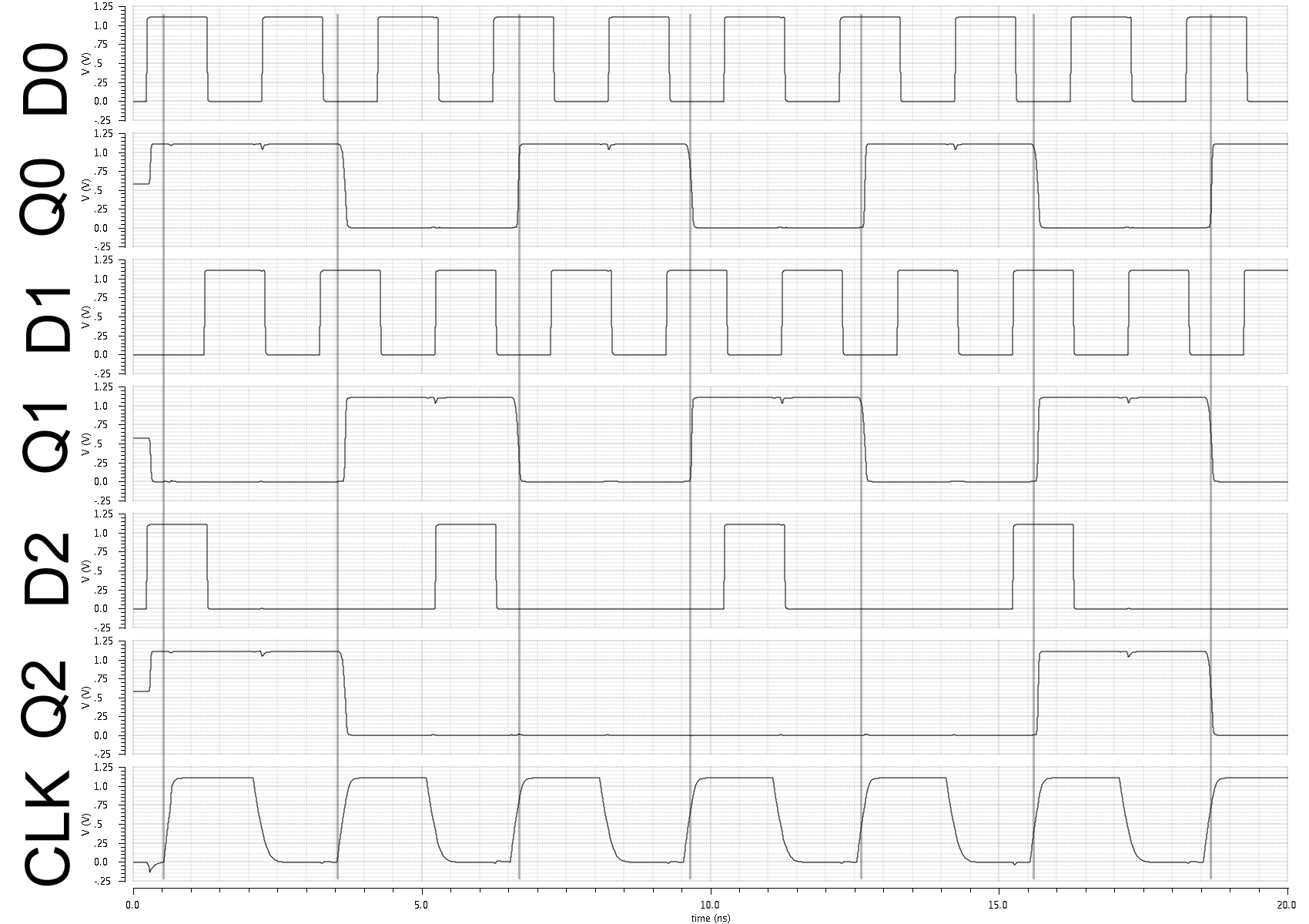
Since the 10th bit is 0, the 11th bit will be 0 when there is 1 shift (00), the 12th bit will be 0 when 2 shifts (01), the 13th bit will be 0 when 3 shifts (10) and the 14th bit will be 0 when 4 shifts (11). Then 15 and 16 will always be 1 since the previous 4 bits are all 1s.

Figure 14 – Register Simulation 1



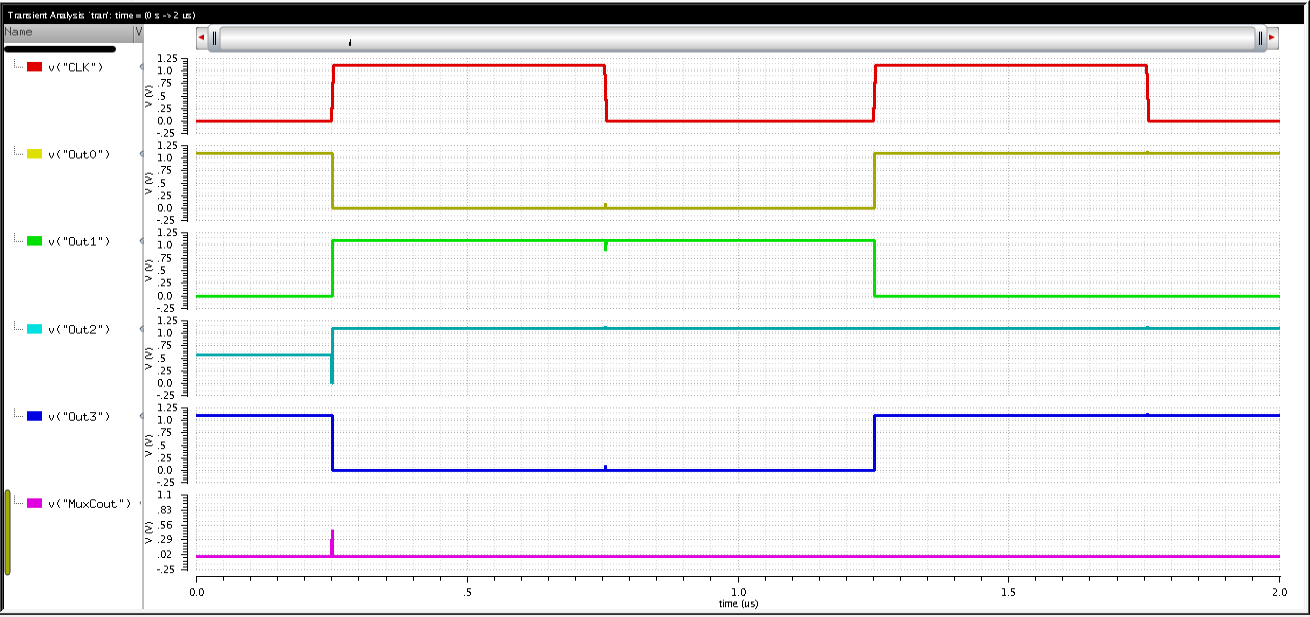
The figure above depicts the operation of a single bit, rising-edge register. Note that the output Q changes on the rising edges of the clocks, as expected. D changes more quickly than the clocks in this simulation, although it has been chosen so that there is never a setup or hold time violation in this example. Rising edges have been marked to show the appropriate devices changes occurred at the appropriate time.

Figure 15 – Register Simulation 2



The figure above depicts operation of part of the 16 bit register. This device is essentially just several 1 bit registers in parallel, so a complete simulation of all inputs and outputs was not deemed necessary.

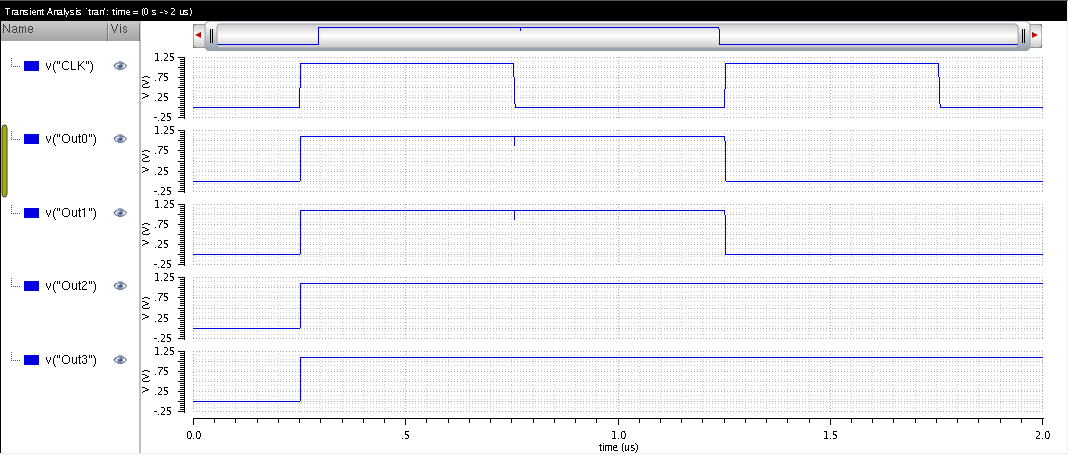
ALU Simulations: For these we only simulated the first 4 bits for simplicity. Also we only simulated one test case to show that it worked since each component was simulated in greater depth individually.

ADD (“000”)

Input A: 000……1010 Input B: 000…….0010 Cin:1

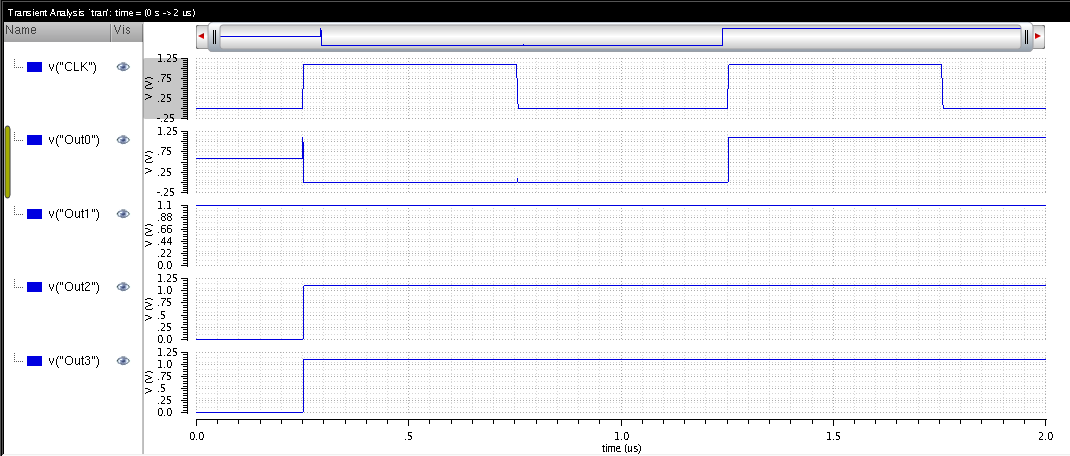
So the result should be 1101 with no carryout which is what is shown.

SHIFT(010)



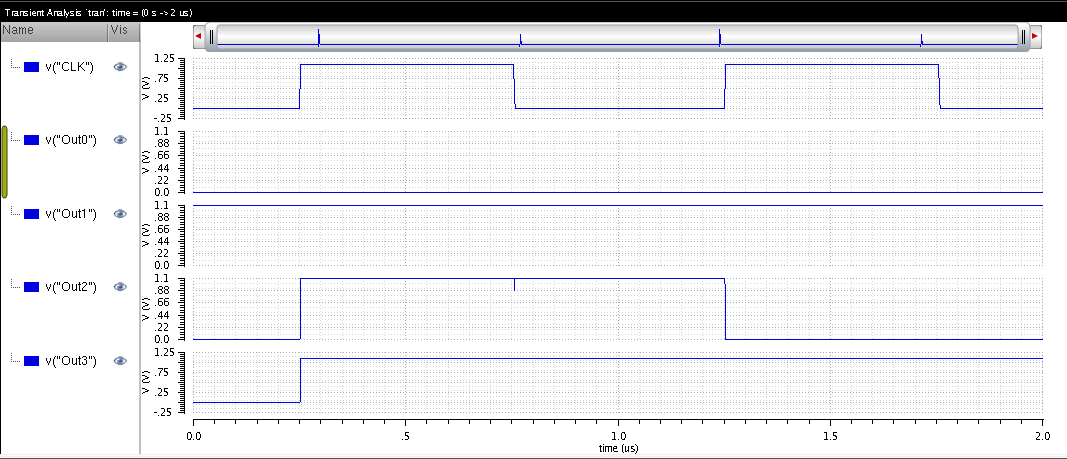
A="0011" and B="xx01” 2 bit shift—Expected: "1100"

OR(100)



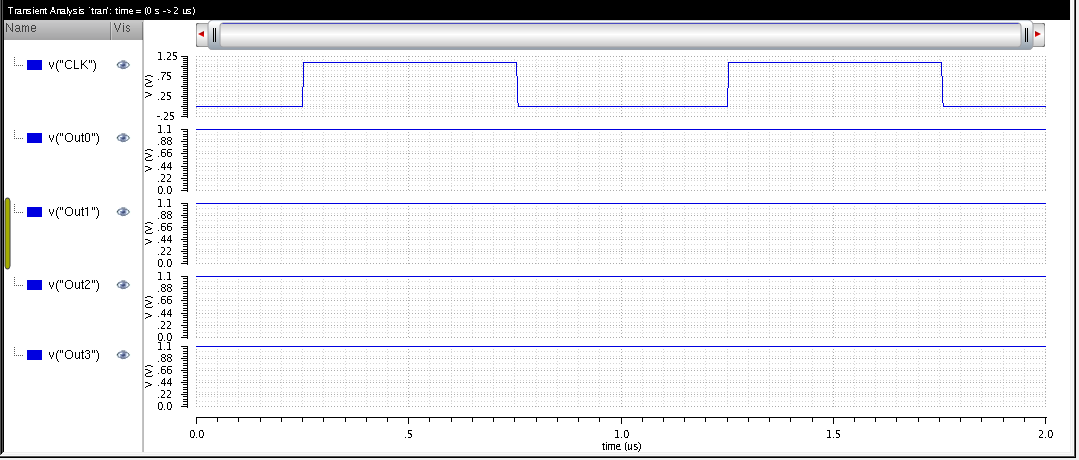
 A="1010", B="0101. Expected: “1111”

PASS(101)



 A="1010", B="0101. Expected: “1010”

NOP(111)



 A="1010", B="0101. Expected: whatever the default value was